Rapport de TP

Intitulé :

TP P2P Web — Développement d'un Serveur Asynchrone et Application FullStack Node.js/Angular

Filière

- SCIENCES DE DONNEES -

Réalisé par :

Hatim Haddioui

Octobre 2025

Sommaire

1. Introduction 3
2. Outils et Environnement de Développement 3
2.1. Outils utilisés 3
3. Étapes de développement (Versions 1 → 6) 4
4. Partie FullStack — Application Node.js + Angular 5
4.1. Diagrammes UML de l'application 5
4.2. Fonctionnalités principales 6
4.3. Communication Client ↔ Serveur 7
4.4. Résultat attendu 7
Conclusion 8

1. Introduction

Ce projet a pour objectif de comprendre et mettre en œuvre les principes de la **communication web en temps réel** à l'aide de **Node.js** et **Angular**.
À travers plusieurs versions successives, nous avons fait évoluer une application simple "Hello World" vers une **plateforme fullstack asynchrone**, capable de gérer plusieurs requêtes simultanément, d'assurer des interactions temps réel et de fournir une interface utilisateur moderne.

2. Outils et Environnement de Développement

2.1. Outils utilisés

Node.js ≥ 18 (LTS) : environnement d'exécution JavaScript côté serveur.

Npm (Node Package Manager) : gestionnaire de dépendances.

Express.js : Framework léger pour la création de serveurs HTTP.

Socket.io : bibliothèque pour la communication en temps réel.

Angular CLI ≥ 17 : Framework front end moderne pour construire le client web.

MongoDB : base de données NoSQL pour la gestion des utilisateurs et fichiers.

Visual Studio Code : environnement de développement.

Postman : pour tester les API HTTP.

Installation et mise en place

- Télécharger et installer Node.js depuis : https://nodejs.org
- Vérification :
  - node -v
  - npm -v
- Initialisation du projet Node.js
  - npm init -y
  - npm install express multer mongoose cors socket.io nodemon
- Lancer le serveur : node index.js

Installation du frontend (Angular)

- Dans le dossier /frontend

```
npm install -g @angular/cli
ng new frontend
cd frontend
npm install
ng serve
```

- L'application Angular sera disponible sur : http://localhost:4200
- Le backend Node.js écoute sur : http://localhost:8888

3. Étapes de développement (Versions 1 → 6)

Version 1 — Serveur minimal "Hello World" [HelloWord.png]

- Objectif : créer un premier serveur HTTP Node.js qui renvoie "Hello World".
- Découverte du module http.

  Test via le navigateur :http://localhost:8888/

Version 2 — Modularisation (server.js + index.js) [version2.png]

- Séparation du code en modules :
  - server.js → création du serveur,
  - index.js → point entrée principal.
- Utilisation du module require() pour importer les fonctions.

Version 3 — Ajout d'un Router [upload.png +start.png]

- Création de router.js pour gérer les différentes routes :
  - /start, /upload, etc.
- Introduction de la logique de routage manuelle.

Version 4 — Handlers [upload.png, find.png, login.png…]

- Ajout du fichier requestHandlers.js contenant les fonctions associées aux routes.
- Exemple : start, upload, show, find.
- Réponse selon le chemin demandé par le client.

Version 5 — Réponse synchrone (anti-pattern) [upload.png…]

- Test du comportement bloquant des fonctions synchrones
- Observation : le serveur devient lent si un traitement long est effectué.
- Correction : remplacement par des appels **asynchrones**.
- Remarque : une erreur s'est produite à cause de localhost/start/.

Version 6 — Réponse asynchrone

- Passage à la gestion asynchrone des requêtes avec des callbacks et fs.readFile().
- Meilleure performance et non-blocage du serveur.
- Fin de la partie Node.js pure.

4. Partie FullStack — Application Node.js + Angular

4.1. Diagrammes UML de l'application

Diagramme de cas d'utilisation

Ce diagramme montre les principales interactions entre l'utilisateur et l'application **P2P Web** à travers l'interface Angular et le serveur Node.js.

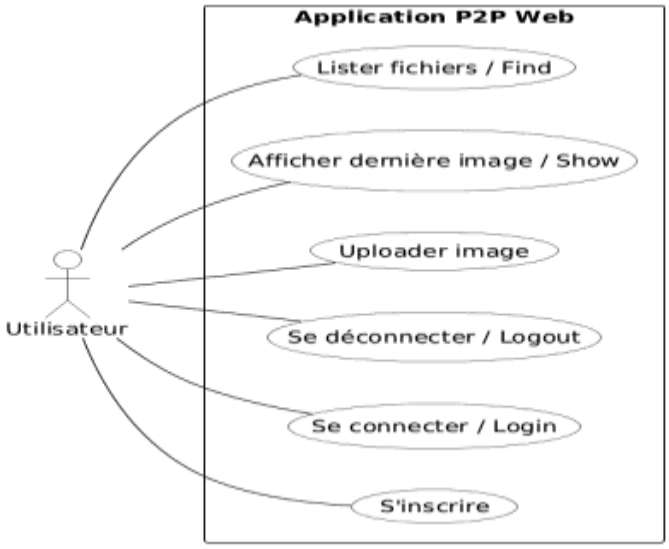
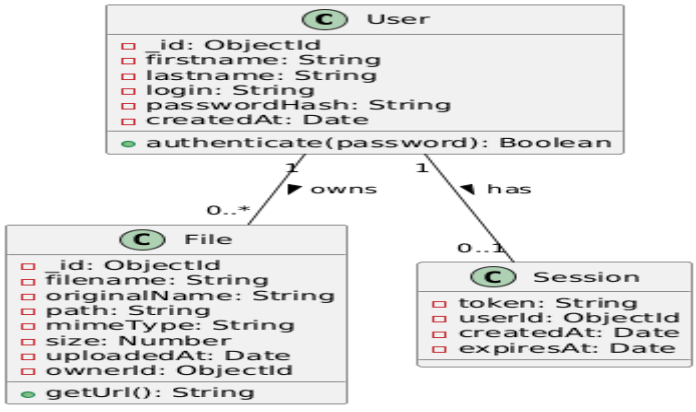


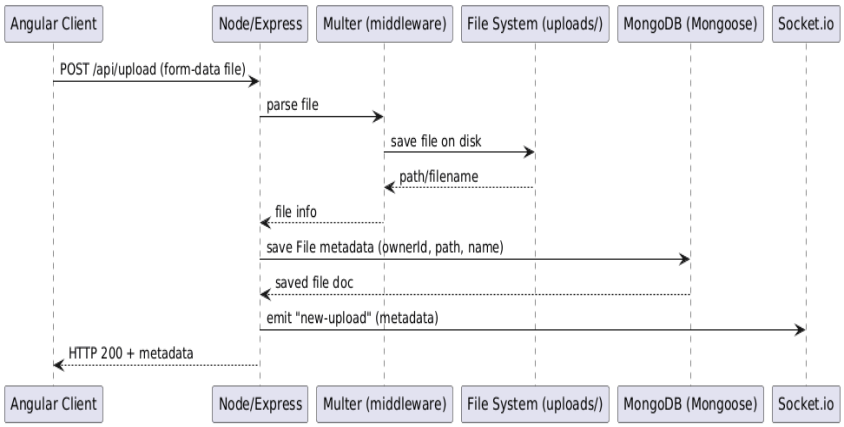
Diagramme de classes

Le diagramme de classes illustre la structure interne de l'application, en représentant les principales entités manipulées côté serveur Node.js et leur relation avec la base de données MongoDB.



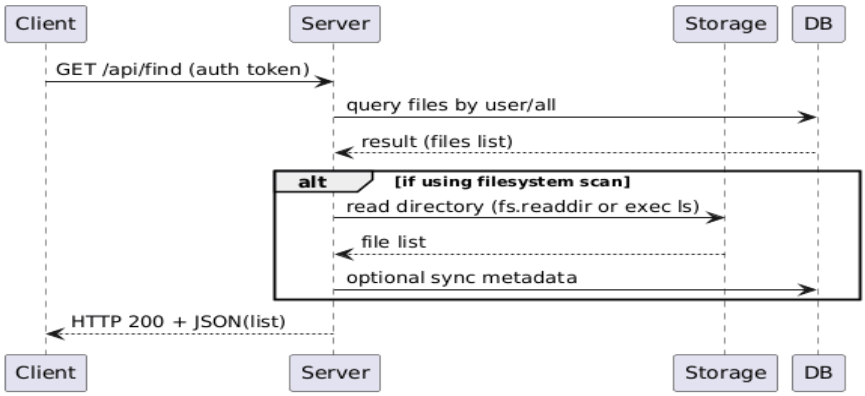
Description du diagramme de séquence — Upload

Ce diagramme illustre le flux complet d'un envoi de fichier (upload) dans une application web basée sur Angular (frontend) et Node.js/Express (backend), avec intégration de Multer, MongoDB, et Socket.io pour la communication temps réel.



Description du diagramme de séquence — Consultation des fichiers (Find Files)

Ce diagramme illustre le processus de récupération des fichiers enregistrés par un utilisateur (ou de tous les fichiers selon le rôle) dans une application Node.js avec un client web (ex. Angular).



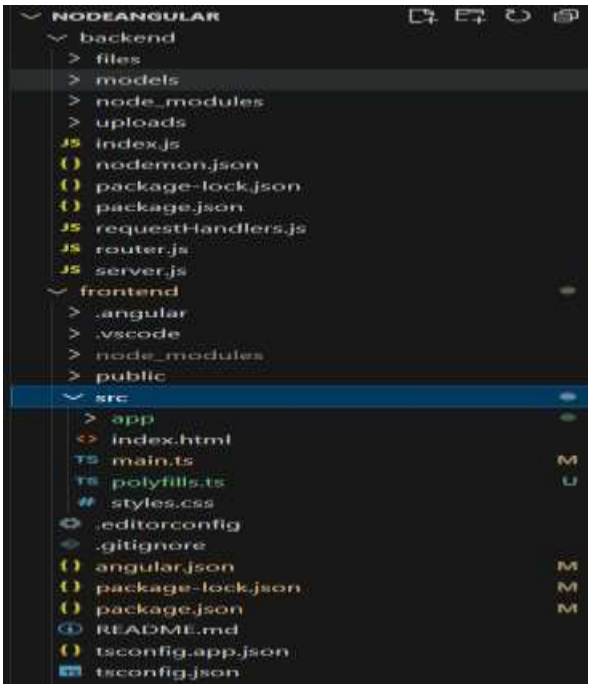
4.2. Fonctionnalités principales

- **Inscription / Connexion des utilisateurs** (avec users.js et sessions.js)
- **Upload de fichiers (images)** avec multer
- **Affichage et liste des fichiers uploadés**
- **Notifications en temps réel** avec Socket.io
- **Frontend Angular** pour interaction fluide et responsive

4.3. Communication Client ↔ Serveur

- Le frontend Angular envoie les requêtes HTTP vers le backend Express.
- Les événements temps réel (upload, suppression, etc.) sont gérés par Socket.io.
- MongoDB stocke les utilisateurs et les informations de fichiers.

Structure finale du projet



Concepts abordés

- Programmation orientée événements (Event-driven)
- Routage HTTP personnalisé
- Gestion asynchrone avec callbacks et promises
- Upload de fichiers avec multer
- Stockage et récupération MongoDB via mongoose
- Notifications en temps réel avec Socket.io
- Intégration frontend Angular avec API REST Node.js

4.4. Résultat attendu

Le résultat final de ce TP est une **application web complète** basée sur l'architecture **fullstack Node.js + Angular**, permettant la communication en temps réel entre le client et le serveur.

Pour visualiser toutes les interfaces et le code associé, voir le dossier complet : TP2/NodeAngular

Conclusion

Ce TP nous a permis de passer d'un simple serveur Node.js à une application web complète et moderne basée sur Node.js et Angular. Nous avons appris à gérer les routes, les requêtes asynchrones, la communication en temps réel et l'intégration d'un frontend dynamique. En résumé, ce projet nous a aidés à mieux comprendre le fonctionnement d'une architecture fullstack et la communication client-serveur en temps réel.